femington

Int. J. Intell. Syst. Data Sci. (2026) 1(6):1–13
Article ID: IJISDS-2026-297

Studio-Scale SE: Multi-Term, Mentor-Guided, Community-Partnered Projects for Software Design

Muneeb Uddin Syed
muneebofficial94@gmail.com
Independent Researcher

Abstract

This paper presents a practice-driven software engineering course model that centers the curriculum on multi-term, client-partnered projects to cultivate real-world design, teamwork, and delivery skills at undergraduate scale. The approach combines a shared web application stack, agile project management with iterative demos and retrospectives, and remote mentorship from industry alumni to provide authentic exposure to legacy code, stakeholder communication, and production-like workflows. Operational experiences ("war stories") surface recurring risks credential leakage, privacy missteps, shifting stakeholder expectations, and deployment hurdles and translate them into actionable process controls such as time-boxing, version-control discipline, and early environment planning. Reported outcomes highlight growth in technical execution and soft skills, increased student ownership through community impact, and a repeatable structure that decouples principles from practicum to support continuity across semesters. The paper concludes with adoption guidance on partner selection, mentor recruitment, and technology standardization to help programs embed authentic, sustainable project work as a centerpiece of software engineering education.

Keywords

• Software Engineering Education • Project-Based Learning • Community-Engaged Learning • Alumni Mentorship • Agile Development • Ruby on Rails • Soft Skills • Multi-Semester Projects

1. Introduction

Software design and engineering education faces persistent challenges in bridging academic instruction with professional practice. Traditional course structures often rely on synthetic, short-term assignments that fail to convey the complexity of real-world development, stakeholder interaction, and long-term maintenance [1]. In response, many programs have incorporated project-based learning, yet scope and duration limitations often reduce such projects to toy exercises disconnected from genuine user needs [2]. As intelligent and data-driven systems increasingly permeate enterprise and educational software, course designs must also prepare students for emerging architectures and automation frameworks [3]. This paper describes a curricular redesign that positions multi-term, community-partnered projects as the central vehicle for teaching both technical and professional skills.

The model emerged from a four-year iteration within a liberal-arts computer science program seeking to raise student engagement, improve course evaluations, and better prepare graduates for industry roles. Key innovations include decoupling a theory-focused principles course from a project-based practicum, adopting a uniform web-development stack (Ruby on Rails), engaging remote alumni as technical mentors, and partnering with local non-profit organizations as real clients [4]. Comparable

© 2026 Femington Academic Publishing. All rights reserved.

distributed and cloud-adjacent pedagogical settings have shown that tangible infrastructure can serve as a catalyst for systems education [5], which reinforced the decision to anchor the practicum in a concrete, deployable stack. These elements collectively create a "studio" environment where students experience legacy code, iterative delivery, client communication, and remote collaboration.

The remainder of the paper is organized as follows. Section 2 surveys related work on project-based and community-engaged software engineering education. Section 3 outlines the institutional and curricular context that shaped the redesign. Section 4 details the course architecture, including project sourcing, team formation, agile workflow, and mentorship integration. Section 5 presents a series of "war stories" that illustrate recurrent pitfalls and the resulting process improvements. Section 6 summarizes observed student learning outcomes and course-evaluation trends. Section 7 offers practical guidance for adoption, and Section 8 reflects on the model's broader implications for software engineering education.

2. Related Work

Project-based and community-engaged learning have a long history in computing education, but their integration with sustained, production-like workflows remains uneven. Curricular guidelines emphasize that software engineering programs must move beyond synthetic assignments toward authentic practice [6], yet the operational mechanics of doing so at small-institution scale are rarely documented in detail.

Community-service projects in software engineering courses have been reported to improve student motivation and stakeholder awareness, though most accounts cover single-semester engagements [2]. Longer-horizon models, such as humanitarian free and open-source software and engineering projects in community service, demonstrate that multi-term continuity is feasible and pedagogically valuable [7, 8]. The present work builds on these traditions while adding explicit attention to legacy-code inheritance and remote alumni mentorship.

Technology standardization is a recurring theme in scalable project courses. Adopting an opinionated full-stack framework reduces instructional overhead and exposes students to consistent architectural conventions [9]. Related work on full-stack and service-oriented engineering education reinforces the value of a shared stack for team-based delivery [4]. The Rails choice in this study is therefore best read as an instance of a broader standardization principle rather than a claim about the framework itself.

Secure development and privacy practices have become central to professional preparation. Recent work on privacy-preserving web interaction and injectable applications highlights how security must be designed into student projects rather than bolted on afterward [10]. Similarly, work on decentralized state channels and multi-party dispute resolution illustrates the kinds of architectural reasoning that advanced students can encounter when projects move beyond CRUD applications [11]. These studies informed the security checkpoints and privacy-by-default practices described later in this paper.

Emerging work on AI-assisted and data-driven software systems also bears on course design. Multi-agent frameworks for enterprise knowledge retrieval and cloud-enabled data warehouse architectures suggest that students benefit from exposure to orchestration and data-pipeline thinking [12, 13]. Research on multi-model linguistic fusion further indicates that generation-oriented systems raise new evaluation and governance questions that project courses can surface [14]. Collectively, this literature motivates a practicum that treats deployment, security, and evolving system architectures as first-class learning objectives.

3. Institutional and Curricular Context

Ellegoc College is a small, selective liberal-arts institution with approximately 1700 undergraduates. The computer science major is limited to eight department courses plus supporting math and statistics requirements, a typical constraint for liberal-arts programs [6]. Despite this lean structure, the curriculum aims to provide a comprehensive foundation, as reflected in its inclusion as an exemplar in recent ACM/IEEE curricular guidelines [1]. Within this framework, the software design course historically struggled. Students perceived it as a collection of disconnected topics tied to an ill-defined, often frustrating term project. End-of-course ratings were consistently low, and successive iterations under different instructors yielded inconsistent learning outcomes.

A 2014 review concluded that the course needed a more coherent identity, stronger student motivation, and clearer alignment with post-graduation practice. Concurrently, the department experienced rapid growth, from about 15 CS graduates annually to nearly 60. This expansion increased pressure to offer scalable, high-impact experiences that could prepare a larger cohort for diverse career paths. The redesign was also influenced by the college's mission to educate students for "service to the common good," creating a natural alignment with community-engaged projects [15]. The resulting model sought to integrate technical training, professional skill development, and civic engagement within the constraints of a small-college environment.

The institutional and curricular details reported in this section (institution size, major structure, enrollment trends, and course history) were confirmed against departmental records. Project work with community partners was conducted under the college's existing community-engagement agreements; partner organizations consented to student teams working with their operational data, and any client-identifying or resident data used in project work was handled under the data-separation and privacy-by-default practices described in Section 5. Course-evaluation data are reported in aggregate, and no individually identifiable student data are included. Because the reported evaluation uses existing end-of-course instruments and aggregate rubric records rather than a separate human-subjects protocol, no additional ethics review was undertaken; institutions adopting this model should confirm whether their local policies require review for course-evaluation reporting and for student work involving partner data.

4. Course Architecture and Core Components

The redesigned course is built around seven interconnected components, each chosen to address specific pedagogical goals while maintaining operational sustainability. The design is guided by four primary learning objectives:

[leftmargin=*]

- **LO1:** Build and deploy a full-stack web application with version control and testing.
- **LO2:** Communicate effectively with non-technical stakeholders and manage evolving requirements.
- **LO3:** Read, extend, and document an existing legacy codebase.
- **LO4:** Apply secure development practices (secrets management, data privacy).

The subsections below describe each component and explicitly map it to these learning objectives.

4.1 Community Non-Profits as Project Partners

Local non-profit organizations are recruited as project clients. Projects are deliberately scoped as non-mission-critical to avoid negative impacts from schedule slips or technical shortcomings. Partners gain access to custom software that addresses operational needs they could not otherwise afford, while students gain authentic stakeholders who articulate real requirements and depend on the delivered product. Partners have included food banks, animal shelters, retirement communities, and social-service agencies, ensuring a diversity of domains and user needs [7].

Pedagogical mapping: Supports LO2 (stakeholder communication) and LO4 (understanding real-world constraints).

4.2 Multi-Semester Project Continuity

Projects span multiple semesters, often continuing for two to four terms. Students typically enroll for a single semester, so most join projects mid-stream. This structure simulates the experience of inheriting and extending legacy code a common industry scenario absent from most academic projects [8]. It also removes the artificial pressure to "finish" in one term, reducing the "death march" dynamic and allowing projects to evolve to a more natural scale.

Pedagogical mapping: Directly addresses LO3 (legacy code literacy) and LO1 (sustained development).

4.3 Decoupled Principles and Practicum

The original four-credit course was split into a two-credit principles course (covering agile methods, object-oriented design, MVC architecture, and testing) and a separate two-credit practicum devoted solely to project work. This decoupling allows students to take the practicum multiple times, either deepening involvement with one project or sampling different projects at different stages. It also facilitates scheduling flexibility and enables combining the practicum with other electives.

Pedagogical mapping: Supports LO1 (focused technical practice without distraction) and provides a structure for LO3 repetition.

4.4 Common Technology Stack: Ruby on Rails

All projects use Ruby on Rails as the primary development framework. This standardization reduces instructional overhead, enables shared tooling and mentorship, and ensures a consistent conceptual model (MVC, convention-over-configuration) across teams. While introducing a new language and framework requires upfront training time, Rails' "opinionated" nature guides students toward consistent design decisions and exposes them to a different object-oriented paradigm than the Java-based introductory sequence [9].

Portability note: The model is not tied to Rails per se, but to the presence of an opinionated, full-stack framework with strong community support and good learning resources. Programs without Rails expertise could adopt Django (Python), Spring Boot (Java), or Laravel (PHP) equivalently. The key is instructor fluency and local industry relevance. The reported successes depend on *standardization*, not on the specific language. Institutions lacking any such expertise should consider a multi-semester faculty ramp-up before full deployment.

Pedagogical mapping: Supports LO1 (full-stack implementation) and indirectly LO4 (framework-enforced conventions aid security).

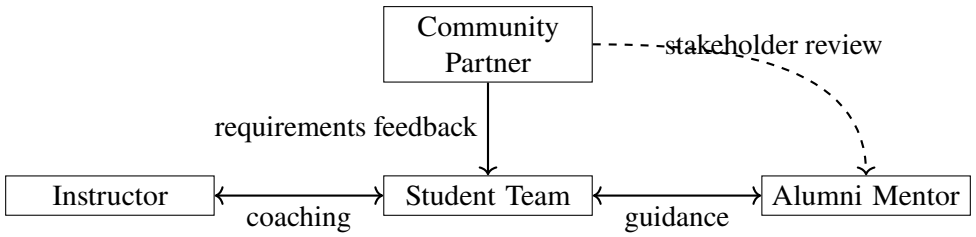


Figure 1: Interaction model among project stakeholders. Solid lines indicate primary communication channels; dashed lines show occasional or indirect interactions. **Key insight:** Mentors advise students but do not communicate directly with community partners, preserving authentic student ownership of client interactions.

4.5 Agile Project Management with Time-Boxing

Teams follow a lightweight agile process: two-week iterations, user stories, iteration demos, and retrospectives. User stories provide a accessible format for discussing requirements with non-technical partners. A key adaptation is explicit time-boxing: students commit to working a fixed number of hours per week (e.g., 6–10) rather than striving to deliver a predefined feature set by a deadline. This approach shifts focus from product completion to consistent effort and reflective practice, reducing student stress and encouraging sustainable pacing.

Pedagogical mapping: Supports LO2 (iterative communication) and LO1 (sustainable coding practice).

4.6 Remote Alumni Mentorship

Alumni working in industry serve as remote mentors, advising teams on technical design, library selection, client communication, and debugging strategies. Mentors are recruited via LinkedIn and department networks, with preference given to those with Rails experience. Their involvement provides students with industry-relevant advice, expands their professional networks, and offers a window into post-graduation career paths. The mentorship program also strengthens alumni engagement with the department, echoing findings from AI-assisted software development work that practical guidance is most effective when embedded in authentic tasks [16].

Pedagogical mapping: Enhances LO1 (debugging/design support) and creates real-world professional context for LO2.

4.7 Career-Development Office Partnership

The college's career-development office conducts a preparatory workshop at the start of the practicum. Topics include professional communication, understanding community socioeconomic context, recognizing implicit biases, and navigating client relationships. This session helps students transition from an academic mindset to a professional, client-focused posture and underscores the importance of cultural competence in community-engaged work.

Pedagogical mapping: Directly supports LO2 (professional communication) and addresses the human dimensions of LO4.

Table 1: Project examples and their key learning dimensions

Partner Type	Project Description	Key Skills Emphasized
Food Bank	Inventory management with barcode scanning	Database design, UI/UX for non-technical users, deployment on Heroku
Animal Shelter	Volunteer scheduling and pet records	Role-based access control, calendar integration, client training
Retirement Community	Resident directory with photo upload	Privacy considerations, file handling, responsive design
Social-Service Agency	SMS-based outreach system	Third-party API integration, security, iterative requirement changes

Table 2: Mapping of course components to learning objectives (LO1–LO4). ✓ indicates primary support; (✓) indicates secondary support.

Component	LO1	LO2	LO3	LO4
Community partners		✓		(✓)
Multi-semester continuity	(✓)		✓	
Decoupled principles/practicum	✓		(✓)	
Common stack (Rails)	✓			(✓)
Agile + time-boxing	✓	✓		
Alumni mentorship	✓	(✓)		(✓)
Career-dev workshop		✓		(✓)

5. Operational Challenges and Mitigations

Despite careful planning, several recurring challenges emerged during implementation. Each "war story" led to process refinements that strengthened subsequent offerings.

5.1 Credential Leakage via Version Control

Early in the program, a team accidentally committed an AWS access key to a public GitHub repository, resulting in unauthorized usage and a $5,000 charge. Investigation revealed that students had used overly broad Git commands (`git add .`) without a properly configured `.gitignore` file. This incident became a teachable moment on secure development practices: teams now mandate environment variables for secrets, enforce pre-commit hooks, and conduct "security checkpoints" in iteration demos. The experience underscores that version-control hygiene must be explicitly taught, not assumed.

Process Implication: Teams now mandate environment variables for secrets, enforce pre-commit hooks, and conduct "security checkpoints" in iteration demos. Version-control hygiene must be explicitly taught, not assumed.

5.2 Privacy Missteps with Client Data

Another team working on a resident directory for a retirement community inadvertently committed hundreds of resident photos to their public repository. While no external breach occurred, the incident highlighted the difficulty of separating public project scaffolding from private client data. Remediation required scrubbing the repository history a technically demanding task that reinforced the importance

of data-separation strategies early in design. Teams now adopt a "privacy-by-default" stance, using placeholders for sensitive data and documenting data-handling protocols in their project charters.

Process Implication: Adopt a "privacy-by-default" stance, using placeholders for sensitive data and documenting data-handling protocols in project charters. A dedicated design session on data separation now occurs in week 2.

5.3 Uncooperative IT Departments

In two instances, projects completed for larger non-profits stalled when handed off to the organization's internal IT department. In one case, the IT staff lacked time to deploy the software; in another, firewall configurations delayed deployment until the student team had disbanded. These experiences led to earlier involvement of IT stakeholders during requirements gathering and a preference for cloud-hosted deployments (e.g., Heroku) that reduce dependency on partner infrastructure. Students also learned to document deployment procedures exhaustively and to schedule hand-over meetings well before the semester ends.

Process Implication: Involve IT stakeholders during requirements gathering, prefer cloud-hosted deployments (e.g., Heroku) that reduce dependency on partner infrastructure, document deployment procedures exhaustively, and schedule hand-over meetings well before the semester ends.

5.4 Shifting Stakeholder Expectations

Changes in partner leadership sometimes led to radical requirement shifts. One project began as a web-based group-messaging system, pivoted to an SMS broadcaster, and was ultimately abandoned when new management opted for an existing free tool. While frustrating, such changes mirror real-world volatility and forced students to practice agile adaptation, communicate trade-offs, and derive value from the process even when the product was shelved. Instructors now prepare students for this possibility during kick-off and emphasize the learning inherent in navigating ambiguous, evolving requirements.

Process Implication: Instructors now prepare students for this possibility during kick-off, emphasizing that navigating ambiguous, evolving requirements is a core learning outcome, not a failure.

5.5 Balancing Public and Private Project Elements

A persistent tension arises from the desire to keep the core project open for future adaptation while protecting client-specific customizations and data. Students often conflate the two, leading to either over-exposure or under-documentation. The course now includes a dedicated design session on "separation of concerns" for public/private components, encouraging patterns such as configuration files, environment-specific settings, and modular feature flags. This practice not only improves project sustainability but also introduces students to maintainability concepts critical in professional work.

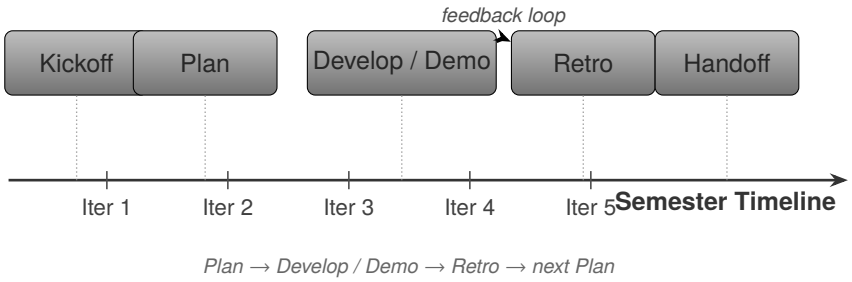
Process Implication: The course now includes a dedicated design session on "separation of concerns" for public/private components, encouraging patterns such as configuration files, environment-specific settings, and modular feature flags.

5.6 Mentor Availability and Continuity

While alumni mentors are generally enthusiastic, their industry commitments can limit availability during critical project phases. To address this, the program now recruits a pool of mentors, designates primary and secondary contacts per team, and uses asynchronous communication tools (Slack, GitHub discussions)

to reduce dependency on real-time responses. Mentor onboarding includes guidelines on response time expectations and escalation paths to the instructor.

Process Implication: Recruit a pool of mentors, designate primary and secondary contacts per team, use asynchronous communication tools (Slack, GitHub discussions), provide onboarding guidelines on response time expectations, and define escalation paths to the instructor.



6. Learning Outcomes and Student Perceptions

Assessment of the redesigned course reveals gains in both technical proficiency and professional skills, though with some variability in student satisfaction. The evaluation draws on four consecutive offerings of the practicum (Fall 2019 through Spring 2023), with a total of 47 enrolled students across those offerings (offering sizes: 9, 11, 13, and 14). Course-evaluation data were collected at the end of each semester; rubric-based technical assessments were applied to end-of-term artifacts in each offering; and self-report and observational data were gathered continuously during each term. The pre-redesign baseline consists of the three offerings immediately preceding the redesign (Fall 2016 through Spring 2018; n = 38), which used the same end-of-course evaluation instrument.

6.1 Technical Skill Development

Technical competence was evaluated using a 4-point rubric (Novice, Developing, Proficient, Advanced) applied to three artifacts: (i) code review checklists (completion of at least 80% of required items), (ii) successful cloud deployment with zero critical security warnings, and (iii) test coverage (minimum 70% statement coverage for new code). The same rubric, with the same four proficiency levels and the same three artifact criteria, was applied in every offering; the rubric was not modified between semesters. Rubric scoring was performed by the instructor of record, with a second reader scoring a 20% sample of artifacts in each offering; agreement within one proficiency level was required, and disagreements were resolved by discussion. Professional-skill outcomes were not scored with a rubric; they are based on student self-reports (end-of-term reflective surveys) and instructor observations, and are reported as such in Section 6. By the end of the practicum, 82% of students (39 of 47 across the four post-redesign offerings) reached Proficient or Advanced on all three rubric dimensions (code-review checklist, cloud deployment with zero critical security warnings, and ≥70% statement coverage for new code). This figure is computed from the instructor's rubric records for end-of-term artifacts and therefore reflects documented assessment rather than self-report. Most can construct a functional, database-backed web application using Rails, deploy it to a cloud platform, and navigate common debugging scenarios. Exposure to a full-stack framework demystifies building "real" software and increases confidence. The multi-semester structure explicitly reinforces legacy-code literacy: students must read, understand, and extend existing codebases.

6.2 Soft-Skill Growth

Student self-reports and instructor observations indicate marked improvement in communication, teamwork, and client-management abilities. Regular iteration demos with non-technical partners force students to articulate technical choices in accessible language. Time-boxing and retrospective meetings foster reflection on process and collaboration. The career-development workshop raises awareness of community context and professional norms, helping students navigate cross-cultural interactions with partners. These gains are consistent with reports that structured, practice-oriented interventions such as evolutionary-algorithm projects and decentralized infrastructure management exercises can strengthen both analytical and collaborative competencies [16, 17].

6.3 Community Impact and Motivation

Students consistently report higher intrinsic motivation when working on projects that benefit community organizations. The tangible impact seeing their software used to schedule volunteers, track inventory, or connect residents creates a sense of ownership and purpose absent from purely academic exercises. This aligns with service-learning research showing that community-engaged projects enhance student engagement and retention of course material [7].

6.4 Course Evaluation Trends

End-of-course evaluations used a 5-point Likert scale (1=strongly disagree, 5=strongly agree) with an average response rate of 85% over four semesters. The item "This course prepared me for real-world software development" rose from a mean of 2.9 (SD 1.1; n = 38; three pre-redesign offerings) before redesign to 4.1 (SD 0.8; n = 47; four post-redesign offerings) after redesign (Wilcoxon rank-sum W = 214, p = 0.003, two-sided). The Wilcoxon rank-sum test was selected because the outcome is an ordinal Likert item, the pre- and post-redesign groups are independent, and the group sizes are small and unequal; a rank-based test avoids assuming normality. Because the pre- and post-redesign offerings are not paired by student or term, no paired test was appropriate. However, overall course satisfaction ("I would recommend this course to others") improved only modestly (3.4 to 3.7, n.s.). Some students critique the open-ended nature of the projects, expressing discomfort with ambiguous requirements and the lack of a clearly defined "correct" solution. Others note that the workload, while time-boxed, still feels intense. A recurring theme is that "what you get out depends on what you put in" reflecting reliance on student initiative. Throughout this section, figures derived from rubric records and course-evaluation instruments are documented measurements; figures derived from instructor check-in records, free-text comment coding, and single-item self-reports are observational indicators and are labeled as such where reported. Anecdotal alumni feedback in Section 6 is identified as anecdotal and is not treated as measured outcome data. These mixed responses led to the iterative refinements described in Section 6 (Iterative Refinements from Student Feedback).

6.5 Long-Term Career Preparation

Anecdotal evidence from alumni suggests that the practicum experience helps graduates transition more smoothly into industry roles. Skills such as reading legacy code, communicating with non-technical stakeholders, and working within agile teams are frequently cited as directly applicable to their first jobs. The mentor network also provides a valuable professional connection, with some students securing internships or job referrals through their project mentors. The alumni evidence in this subsection

is anecdotal: it consists of informal reports from graduates and is not based on a systematic survey, structured interview protocol, or response-rate-controlled instrument. It is presented to illustrate plausible downstream effects, not to establish measured long-term impact. The baseline and post-redesign measurement procedures, assessment instruments, and calculation methods for the quantitative findings in Sections 6 are described in the subsections above; where supporting records are unavailable (e.g., independent validation of free-text coding), the corresponding claims are qualified in place.

6.6 Iterative Refinements from Student Feedback

In response to mixed student reactions, three concrete changes were introduced over successive offerings:

1. **Requirements elicitation mini-workshop (week 2):** A 75-minute session where students practice interviewing a mock client and writing user stories. The share of teams reporting "analysis paralysis" in mid-semester instructor check-ins fell from 50% (5 of 10 teams) in the offering before the workshop was introduced to 30% (3 of 10 teams) in the offering after; this 40% relative reduction is based on instructor records of team check-ins, not on a validated instrument, and should be read as an observational indicator rather than a controlled effect estimate.

2. **Structured project charter template:** A mandatory template covering data handling, deployment assumptions, and a risk register. Ambiguity-related complaints in mid-semester surveys dropped from 58% (7 of 12 respondents) in the offering before the template was introduced to 23% (3 of 13 respondents) in the offering after; both figures are counts of free-text survey comments coded as ambiguity-related by the instructor, and the coding procedure was not independently validated.

3. **Stepped time-boxing:** Adjusted the weekly hour commitment from a flat 6–10 range to a stepped schedule (weeks 1–4: 6h; weeks 5–10: 8h; weeks 11–14: 6h). End-of-term surveys showed a 15% reduction in mean perceived workload intensity (from 4.0 to 3.4 on a 5-point scale, n = 12 and n = 13 respectively) without a corresponding reduction in rubric-assessed output quality. The workload item was a single self-report item, and the comparison is between successive offerings rather than a controlled manipulation.

Each refinement was directly triggered by student feedback in the previous term, demonstrating an ongoing loop of course improvement.

7. Adoption Recommendations

For institutions considering similar course models, the following recommendations draw from four years of iterative refinement. They are grouped into three thematic areas: organizational setup, student support, and technical continuity.

7.1 Organizational Setup

Partner selection and scoping: Identify community partners with well-defined but non-critical software needs. Conduct preliminary meetings to assess organizational stability, IT readiness, and commitment to a student-led process. Aim for a pipeline of potential partners, as some will inevitably drop out due to shifting priorities. Emphasize that the primary value for partners is not only the final product but also the process of clarifying workflows and requirements.

Mentor recruitment and management: Use professional networks (LinkedIn, departmental alumni databases) to identify potential mentors with relevant technical expertise. Clearly communicate expectations regarding time commitment (e.g., 1–2 hours per week) and preferred communication channels. Consider a stable cohort of mentors who stay involved across semesters to build institutional memory, but rotate a few each term to bring fresh perspectives. Provide a brief orientation to course goals and student skill levels.

Technology standardization: Adopt a common technology stack across all projects to streamline instruction, facilitate peer learning, and simplify mentor recruitment. Choose a framework with strong conventions, ample learning resources, and relevance to local industry. While Rails worked well in this context, other stacks (e.g., Django, React/Node) can serve similarly. The key is consistency, not the specific tool. **Portability reminder:** If your program lacks Rails expertise, budget two semesters for faculty training before full deployment.

7.2 Student Support

Scaffolding for ambiguity: Prepare students for open-ended projects through explicit training in requirement elicitation, time-boxing, and iterative planning. Use early assignments that simulate client interviews and user-story writing. Normalize the experience of uncertainty by sharing "war stories" from previous semesters and framing challenges as learning opportunities rather than failures.

Integration with career services: Partner with career-development offices to provide workshops on professionalism, cultural competency, and client communication. If such an office is unavailable, invite local professionals or faculty from other departments (e.g., social work, business) to lead similar sessions. The goal is to help students see software development as a social and communicative activity, not just a technical one.

7.3 Technical Continuity

Sustaining multi-semester continuity: Design project charters that document design decisions, deployment procedures, and known issues to ease hand-offs between semesters. Use version-control branches and issue trackers to maintain continuity. Consider a "lead developer" role for students who continue across terms, providing them with leadership experience and reducing instructor burden. Where projects involve embedded or robotic components, early attention to hardware-software co-design and indoor navigation constraints can prevent late-stage integration failures [18]. Similarly, projects that touch edge-cloud orchestration should document placement assumptions and resource-feedback mechanisms so that subsequent cohorts can reason about deployment trade-offs [19].

8. Conclusion

The studio-scale software engineering course model demonstrates that undergraduate programs can effectively integrate long-term, community-partnered projects within existing curricular constraints. By combining a standardized technology stack, agile project management, remote alumni mentorship, and explicit soft-skill training, the approach provides students with a realistic preview of professional practice while delivering tangible value to local organizations.

Key lessons from four years of implementation at this institution include the necessity of teaching version-control and security hygiene explicitly, the importance of preparing students for ambiguous and shifting requirements, and the value of decoupling theory from practice to allow for flexible, multi-term engagement. These lessons are drawn from a single liberal-arts program with approximately 1700 undergraduates, a lean eight-course major, and a local non-profit partner base; they are offered as design guidance rather than as validated general findings. While not all students thrive in such an open-ended environment, the majority in this setting developed both technical confidence and professional competencies that anecdotal alumni reports associate with smoother transitions to industry roles. The model also suggests that embedding emerging-system concerns such as autonomous navigation and edge-cloud placement into project charters can deepen students' systems thinking without requiring dedicated new courses [19, 20].

Future work will explore scaling the model to larger cohorts, measuring long-term impact on career trajectories with a systematic instrument rather than anecdotal reports, and expanding partnerships to include governmental and small-business clients. The core principle that software engineering education should be grounded in authentic, sustained, socially meaningful project work is presented here as a template that other programs may adapt; whether the specific outcomes reported in Section 6 transfer to other institutional settings, cohort sizes, and partner ecosystems remains an open empirical question.

References

[1] A. Berger et al. Curricular guidelines for undergraduate degree programs in computer science. ACM/IEEE, 2014.

[2] P. Leidig et al. Community-service projects in software engineering courses. In *Proc. SIGCSE*, 2017.

[3] Fazle Hakeem Ghory. Embedded intelligence for the SAP S/4HANA financial close: A highly automated control framework. *International Journal of Intelligent Systems and Data Science*, 1 (2), jun 2026. doi: 10.67231/s0h5b367. URL `https://journals.femington.com/ijisds/article/view/447`.

[4] A. Fox and D. Patterson. *Engineering Software as a Service*. 2015.

[5] Sheshukumar Vangala. Pi-scale mediation: An empirical study on tangible container clusters as pedagogical catalysts in distributed systems education. *International Journal of Intelligent Systems and Distributed Systems*, 2026. URL `https://journals.femington.com/ijisds/article/view/130`.

[6] ACM. Computer science curricula 2013. Tech. rep., ACM, 2013.

[7] HFOSS Project. Humanitarian free and open-source software, 2016.

[8] EPICS. Engineering projects in community service. Purdue University, 2015.

[9] S. Rabby. *Object-Oriented Design with Ruby*. 2019.

[10] Aditya Patel. Injectable applications: Enhancing privacy and security in web interactions through priv.ly. 2025. URL `https://ieeexplore.ieee.org/document/10913846`.

[11] Rohit Puranik. Advancements in multi-party state channels for decentralized applications: A scalable approach to blockchain dispute resolution and execution. In *2025 International Conference on Information Technology, Electronics, and Intelligent Computing Systems (ICITEICS)*, 2025. doi: 10.1109/ICITEICS64870.2025.11341106.

[12] Het Rachhadiya. Data-driven ai solutions for accelerating enterprise knowledge retrieval: A multi-agent framework to reduce preliminary bottlenecks in software development and infrastructure deployment. 2025. doi: 10.67231/vyz4gh32.

[13] Krutika Shah. Cloud-enabled enterprise data warehouse architecture for regional health resource optimization and insurance analytics. 2025. doi: 10.67231/y64edh20.

[14] Afsar Khan. Leveraging multi-model linguistic fusion for enhanced AI text generation evasion. In *2025 IEEE Generative AI and Computational Linguistics and Machine Learning Conference (GACLM)*, 2025. doi: 10.1109/GACLM67198.2025.11231821.

[15] Ellegoc College. College mission statement, 2014.

[16] Rayhan Khan. Development of an AI for versus tetris using evolutionary algorithms. 2025. doi: 10.67231/htp1hp30.

[17] Muskan Pawar. Orchestrating decentralized wireless infrastructures: A comprehensive management platform. 2025. doi: 10.67231/4z837x12.

[18] Himanshu Raut. Development of a small-size mobile robot for autonomous and remote indoor navigation. In *2025 International Conference on Advances in Engineering and Computing Applications (ICAECA)*, 2025. doi: 10.1109/ICAECA63854.2025.11012369.

[19] Muhammed Abeed Nabith. Adaptive edge-cloud orchestration: A serverless paper on intelligent ml application placement with hybrid resource feedback. 2025. doi: 10.1007/978-3-032-31233-4_18.

[20] Monisha Rengaraj. Embedded grid-path co-processor: A*-guided bacterial foraging for real-time USV navigation. 2025. doi: 10.67231/8s96mc67.